Package ‘mipfp’

October 13, 2022

Type Package

Title Multidimensional Iterative Proportional Fitting and Alternative Models

Version 3.2.1

Date 2018-08-29

Author Johan Barthelemy [aut, cre],
Thomas Suesse [aut],
Mohammad Namazi-Rad [ctb]

Maintainer Johan Barthelemy <johan@uow.edu.au>

Description An implementation of the iterative proportional fitting (IPFP), maximum likelihood, minimum chi-square and weighted least squares procedures for updating a N-dimensional array with respect to given target marginal distributions (which, in turn can be multidimensional). The package also provides an application of the IPFP to simulate multivariate Bernoulli distributions.

LazyData yes

Depends cmm, Rsolnp, numDeriv, R(>= 2.10.0)

License GPL-2

URL https://github.com/jojo-/mipfp

NeedsCompilation no

Repository CRAN

Date/Publication 2018-08-29 08:10:07 UTC

R topics documented:

`mipfp-package` *Multidimensional Iterative Proportional Fitting and Alternative Models*

Description

An implementation of several methods for updating an initial N-dimensional array (called a seed) with respect to given target marginal distributions. Those targets can also be multi-dimensional. The procedures are also able to estimate a (multi-dimensional) contingency table (encoded as an array) matching a given set of (multi-dimensional) margins. In that case, each cell of the seed must simply be set to 1.

The package provides the iterative proportional fitting procedure (IPFP), also known as the RAS algorithm in economics and matrix raking or matrix scaling in computer science. Additionnaly several alternative estimating methods to the IPFP are also included, namely the maximum likelihood (ML), minimum chi-squared (CHI2) and weighted least squares (WLSQ) model-based approaches.

The package also includes an application of the IPFP to simulate and estimate the parameters of multivariate Bernoulli distributions.

Finally a function extracting the linearly independant columns from a matrix, hence returning a matrix of full rank is provided.

Details

Package:	mipfp
Type:	Package
Version:	3.2.1
Date:	2018-08-29
Depends:	cmm, numDeriv, Rsolnp, R(>= 2.10.0)
License:	GPL-2

This package provides an implementation of several fitting procedures for updating a N-dimensional array with respect to given target marginal distributions. Those targets can also multi-dimensional. The available methods are listed herehunder.

- The function `Ipfp` provides the iterative proportionnal fitting Procedure.
- Maximum likelihood, minimum Chi-square and weighted least squares approaches are availables in the function ObtainModelEstimates.

The function `Estimate` provides an interface to these two methods. Each of them returns an object of class `mipfp`, but `Estimate` should be the preferred constructor.

The package provides several methods and functions to extract various information from the resulting object such as as the variance-covariance matrix of the estimated cell probabilities or counts using either the Lang's (2004) or the Delta method (Little and Wu, 1991) (`vcov`), the confidence interval of the estimates (`confint`), the comparison of the deviations (`CompareMaxDev`), etc. Note that the functions starting with a lower case are S3 methods for objects of class `mipfp` while the one starting with an upper case are general functions.

The package also includes an application of the IPFP to simulate and estimate the parameters of multivariate Bernoulli distributions, respectively in the functions `RMultBinary` and `ObtainMultBinaryDist`. In addition, the functions `Corr2Odds`, `Odds2Corr`, `Corr2PairProbs`, `Odds2PairProbs` are in turn responsible for converting correlation to odds ratio, odds ratio to correlation, correlation to pairwise probability and odds ratio to pairwise probability.

Finally, auxillary functions are also provided. `expand` expands a multi-dimensional contingency table (stored in `table`) into a data frame of individual recors. `Array2Vector` and `Vector2Array` transforms an array to a vector and vice-versa. `flat` flattens multi-dimensional objects for pretty printing. The function `GetLinInd` extracting the linearly independant columns from a matrix (using QR decomposition) and returning a matrix of full rank is also provided.

Author(s)

Johan Barthelemy and Thomas Suesse.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Bacharach, M. (1965). Estimating Nonnegative Matrices from Marginal Data. *International Economic Review* (Blackwell Publishing) 6 (3): 294-310.

Barthelemy, J., Suesse, T. (2018). mipfp: An R Package for Multidimensional Array Fitting and Simulating Multivariate Bernoulli Distributions. *Journal of Statistical Software, Code Snippets* 86 (2): 1-20, doi: 10.18637/jss.v086.c02.

Bishop, Y. M. M., Fienberg, S. E., Holland, P. W. (1975). *Discrete Multivariate Analysis: Theory and Practice.* MIT Press. ISBN 978-0-262-02113-5.

Deming, W. E., Stephan, F. F. (1940). On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. *Annals of Mathematical Statistics* 11 (4): 427-444.

Fienberg, S. E. (1970). An Iterative Procedure for Estimation in Contingency Tables. *Annals of Mathematical Statistics* 41 (3): 907-917.

Golub, G. H., Van Loan C. F. (2012) *Matrix Computations. Third Edition.* Johns Hopkins University Press.

Lang, J.B. (2004) Multinomial-Poisson homogeneous models for contingency tables. *Annals of Statistics* 32(1): 340-383.

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Little, R. J., Wu, M. M. (1991) Models for contingency tables with known margins when target and sampled populations differ. *Journal of the American Statistical Association* 86 (413): 87-95.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

Stephan, F. F. (1942). Iterative method of adjusting frequency tables when expected margins are known. *Annals of Mathematical Statistics* 13 (2): 166-178.

See Also

`ipfp` for a package implementing the ipfp to solve problems of the form $Ax = b$.

Examples

```
# generation of an intial 2-ways table to be updated
seed <- array(1, dim=c(2, 2))
# desired targets (margins)
target.row <- c(87, 13)
target.col <- c(52, 48)
# storing the margins in a list
target.data <- list(target.col, target.row)
# list of dimensions of each marginal constrain
target.list <- list(1, 2)
# calling the fitting methods
r.ipfp <- Ipfp(seed, target.list, target.data)
r.ml <- ObtainModelEstimates(seed, target.list, target.data, method = "ml")
r.chi2 <- ObtainModelEstimates(seed, target.list, target.data, method = "chi2")
r.lsq <- ObtainModelEstimates(seed, target.list, target.data, method = "lsq")
```

`Array2Vector` *Transforming an array to a vector*

Description

Transform a N-dimensional array a to vector. The transformation is done assuming that the last index of the array moves fastest. For instance, an array a of dimensions (2,2,2) will produce the vector $v = (a_{111}, a_{112}, a_{113}, a_{121}, a_{122}, \ldots, a_{333})$.

Usage

```
Array2Vector(arr)
```

Arguments

`arr`	The array to be transformed.

Value

A vector filled with the data of the input array `arr`.

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

See Also

The inverse transformation is performed with the function `Vector2Array`

Examples

```
# generating an array of dimension (3,3,3)
a <- array(seq(1:27),dim=c(3,3,3))
# transforming it into a vector
v <- Array2Vector(a)
```

coef.mipfp *Extract the coefficients of the estimates from an object of class mipfp*

Description

This method extracts the coefficients of estimates of an mipfp object.

Usage

```
## S3 method for class 'mipfp'
coef(object, prop = FALSE, ...)
```

Arguments

`object`	An object of class `mipfp`
`prop`	If this Boolean is set to `TRUE` then the method will return the estimated probabilities. Otherwise, it will return the estimated counts. Default is `False`.
`...`	Not used.

Value

Coefficients of the estimates extracted from the `mipfp` object `object`.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy `<johan@uow.edu.au>`.

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S.* Wadsworth & Brooks/Cole.

See Also

`coef`.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
```

```
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10 pct sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# estimating a table using ipfp
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data)
# exctracting and printing the coefficient of the estimates
print(coef(r.ipfp))
```

`CompareMaxDev` *Comparing deviations of mipfp objects*

Description

This function compares either the margins errors from different `mipfp` objects or the absolute maximum deviation between a given table and the estimates in the mipfp objects.

Usage

```
CompareMaxDev(list.mipfp = list(), true.table = NULL, echo = FALSE)
```

Arguments

`list.mipfp`	The list produced by the function `Estimate`.
`true.table`	When provided, the estimates contained in the `mipfp` objects in the list `list.mipfp` are compared against this table. It is an optional argument.
`echo`	Verbose parameter. If `TRUE`, the function prints what is being compared. Default is `FALSE`.

Value

A table with as many rows as the number of mipfp objects in `list.mipfp`. Each row details the margins errors or the maximum absolute deviation of one `mipfp` object.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

See Also

The estimation function `Estimate`.

This function is used by `error.margins.mipfp`.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10% sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# applying the different fitting methods
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data,  method = "ipfp")
r.ml   <- Estimate(seed = seed.table, target.list = tgt.list.dims,
                   target.data = tgt.data, method = "ml")
r.chi2 <- Estimate(seed = seed.table, target.list = tgt.list.dims,
                   target.data = tgt.data, method = "chi2")
r.lsq  <- Estimate(seed = seed.table, target.list = tgt.list.dims,
                   target.data = tgt.data, method = "lsq")
# print the maximum absolute deviation between targets and generated margins
CompareMaxDev(list(r.ipfp,r.ml,r.chi2,r.lsq), echo = TRUE)
# compute the maximum absolute deviation between the true and estimated tables
CompareMaxDev(list(r.ipfp,r.ml,r.chi2,r.lsq), echo = TRUE,
              true.table = true.table)
```

`ComputeA` *Computes the marginal matrix A and margins vector m of an estimation problem*

Description

Given a set of marginal target constraints and the dimension of the array X to wich the targets relate to, this function computes the matrix A of full rank and vector m such that

$$A^T \pi = (m, 1)^T$$

where vector m contains all components but one of every target and π is a vector of the (unknow) components of X.

Usage

```
ComputeA(dim.arr, target.list, target.data)
```

Arguments

- `dim.arr` The dimension of the array X to which the margins are applied.
- `target.list` A list of the target margins provided in target.data. Each component of the list is an array whose cells indicates which dimension the corresponding margin relates to.
- `target.data` A list containing the data of the target margins. Each component of the list is an array storing a margin. The list order must follow the one defined in target.list. Note that the cells of the arrays must be non-negative.

Value

A list whose elements are defined below.

- `marginal.matrix` The marginal matrix.
- `margins` A vector containing the margins associated with A.
- `df` The degree of freedom of the problem.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

See Also

`MarginalMatrix`.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10 pct sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# computing the associated marginal matrix and margins vector
res.marg <- ComputeA(dim(seed.table), tgt.list.dims, tgt.data)
print(res.marg)
```

`confint.mipfp` *Computing confidence intervals for the mipfp estimates*

Description

This function computes the (asymptotic) Wald confidence intervals at a given significance level for the estimates of an `mipfp` object generated by `Estimate`.

Usage

```
## S3 method for class 'mipfp'
confint(object, parm, level = 0.95, prop = FALSE, ...)
```

Arguments

`object`	The `mipfp` object containing the estimates.
`parm`	A specification of which estimates are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all estimates are considered.
`level`	The confidence level required.
`prop`	A boolean indicating if the results should be using counts (`FALSE`) or proportion (`TRUE`). Default is `FALSE`.
`...`	Further arguments passed to or from other methods (for instance `vcov.mipfp`).

Details

The confidence interval of the estimates $\hat{X}$, at significance level α is given by

$$\hat{X} \pm z\left(1 - \frac{\alpha}{2}\right) * \hat{\sigma}$$

where $\hat{\sigma}$ is the standart deviations of $\hat{X}$, z and $\alpha = 1 - level$ is the inverse of the cumulative distribution function of the standard normal distribution.

Value

A matrix containing the upper and lower bounds for the estimated counts/probabilities (depending on the value of the `prop` argument).

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Smithson, M. (2002). *Confidence intervals*. Sage Publications.

See Also

confint for the default method to compute confidence intervals for model parameters. Estimate, Ipfp and ObtainModelEstimates to generate the mipfp objects for this function.

Examples

```
# true contingency (2-way) table
true.table <- array(c(43, 44, 9, 4), dim = c(2, 2))
# generation of sample, i.e. the seed to be updated
seed <- ceiling(true.table / 10)
# desired targets (margins)
target.row <- apply(true.table, 2, sum)
target.col <- apply(true.table, 1, sum)
# storing the margins in a list
target.data <- list(target.col, target.row)
# list of dimensions of each marginal constrain
target.list <- list(1, 2)
# using ipfp
res <- Estimate(seed, target.list, target.data)
# computing and printing the confidence intervals
print(confint(res))
```

Corr2Odds *Converting correlation to odds ratio*

Description

For K binary (Bernoulli) random variables X_1, ..., X_K, this function transforms the correlation measure of association C_{ij} between every pair (X_i, X_j) to the odds ratio O_{ij} where

$$C_{ij} = \frac{cov(X_i, X_j)}{\sqrt{var(X_i) * var(X_j)}}$$

and

$$O_{ij} = \frac{P(X_i = 1, X_j = 1) * P(X_i = 0, X_j = 0)}{P(X_i = 1, X_j = 0) * P(X_i = 0, X_j = 1)}.$$

Usage

```
Corr2Odds(corr, marg.probs)
```

Arguments

corr	A $K \times K$ matrix where the i-th row and the j-th column represents the correlation C_{ij} between variables i and j.
marg.probs	A vector with K elements of marginal probabilities where the i-th entry refers to $P(X_i = 1)$.

Value

The function return a list with the correlations and the pairwise probabilities.

`odds`	A matrix of the same dimension as `corr` containing the correlations
`pair.proba`	A matrix of the same dimension as `corr` containing the pairwise probabilities.

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

`Corr2Odds` for converting correlation to odds ratio.

Examples

```
# correlation matrix from Qaqish et al. (2012)
cr <- matrix(c( 1.000, -0.215, 0.144, 0.107,
               -0.215,  1.000, 0.184, 0.144,
                0.144,  0.184, 1.000, 0.156,
                0.107,  0.144, 0.156, 1.000), nrow = 4, ncol = 4, byrow = TRUE)
rownames(cr) <- colnames(cr) <- c("Parent1", "Parent2", "Sibling1", "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# converting correlation to odds ratio and getting pairwise probabilities
or <- Corr2Odds(corr = cr, marg.probs = p)
print(or)
```

`Corr2PairProbs` *Converting correlation to pairwise probability*

Description

For K binary (Bernoulli) random variables X_1, ..., X_K, this function transforms the correlation measure of association C_{ij} between every pair (X_i, X_j) to the pairwise probability $P(X_i = 1, X_j = 1)$, where C_{ij} is defined as

$$C_{ij} = \frac{cov(X_i, X_j)}{\sqrt{(var(X_i) * var(X_j))}}.$$

Usage

```
Corr2PairProbs(corr, marg.probs)
```

Arguments

`corr`	A $K \times K$ matrix where the i-th row and the j-th column represents the correlation C_{ij} between variables i and j.
`marg.probs`	A vector with K elements of marginal probabilities where the i-th entry refers to $P(X_i = 1)$.

Value

A matrix of the same dimension as `corr` containing the pairwise probabilities

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

`Odds2PairProbs` for converting odds ratio to pairwise probability.

Examples

```
# correlation matrix from Qaqish et al. (2012)
corr <- matrix(c( 1.000, -0.215, 0.144, 0.107,
                 -0.215,  1.000, 0.184, 0.144,
                  0.144,  0.184, 1.000, 0.156,
                  0.107,  0.144, 0.156, 1.000),
                  nrow = 4, ncol = 4, byrow = TRUE)
rownames(corr) <- colnames(corr) <- c("Parent1", "Parent2", "Sibling1",
                                      "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# getting the pairwise probabilities
pp <- Corr2PairProbs(cor = corr, marg.probs = p)
print(pp)
```

`error.margins` *Extracts the deviation between every target and generated margin*

Description

This method returns the maximum deviation between each generated and desired margins of the input argument. It corresponds to the absolute maximum deviation between each target margin used to generate the estimates in the mipfp object and the generated one.

Usage

```
## S3 method for class 'mipfp'
error.margins(object, ...)
```

Arguments

`object`	An object of class `mipfp`.
`...`	Further arguments passed to or from other methods. See `CompareMaxDev`.

Value

An array containing the absolute maximum deviations for each margin.

Note

It is an alias for `CompareMaxDev` when only one object is passed to the function and the verbose parameter is set to `FALSE`.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

See Also

The estimation function `Estimate`.

This function relies on `CompareMaxDev`.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
```

```
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10% sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# applying a fitting method
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data,  method = "ipfp")
# print the maximum absolute deviation between targets and generated margins
print(error.margins(r.ipfp))
```

`Estimate` *Update an N-way table given target margins*

Description

This function provides several estimating methods to up multiway table (referred as the seed) subject to known constrains/totals: Iterative proportional fitting procedure (ipfp), maximum likelihood method (ml), minimum chi-squared (chi2) and weighted least squares (lsq). Note that the targets can also be multi-dimensional.

Usage

```
Estimate(seed, target.list, target.data, method = "ipfp", keep.input = FALSE,
          ...)
```

Arguments

`seed`	The initial multi-dimensional array to be updated. Each cell must be non-negative if `method` is `ipfp` or strictly positive when `method` is `ml`, `lsq` or `chi2`.
`target.list`	A list of dimensions of the marginal target constrains in `target.data`. Each component of the list is an array whose cells indicate which dimension the corresponding margin relates to.
`target.data`	A list containing the data of the target marginal tables. Each component of the list is an array storing a margin. The list order must follow the ordering defined in `target.list`. Note that the cells of the arrays must be non-negative.
`method`	An optional character string indicating which method is to be used to update the seed. This must be on of the strings "ipfp", "ml", "chi2", or "lsq". Default is "ipfp".
`keep.input`	A Boolean indicating if `seed`, `target.data` and `target.list` when set to `TRUE`.
`...`	Additionals argument that can be passed to the functions Ipfp and ObtainModelEstimates. See their respective documentation for more details.

Value

An object of class mipfp is a list containing at least the following components:

- **x.hat** — An array with the same dimension of seed whose margins match those specified in target.list.
- **p.hat** — An array with the same dimension of x.hat containing the updated cell probabilities, i.e. x.hat / sum(x.hat).
- **error.margins** — A list returning, for each margin, the absolute maximum deviation between the desired and generated margin.
- **conv** — A boolean indicating whether the algorithm converged to a solution.
- **evol.stp.crit** — The evolution of the stopping criterion over the iterations (if selected method is "ipfp")).
- **solnp.res** — The estimation process uses the solnp optimisation function from the R package Rsolnp and solnp.res is the corresponding object returned by the solver (if selected method is not "ipfp").
- **method** — The selected method for estimation.
- **call** — The matched call.

The will be also added if keep.input has been set to TRUE: seed, target.data, target.list.

Note

It is important to note that if the margins given in target.list are not consistent (i.e. the sums of their cells are not equals), the input data is then normalised by considering probabilities instead of frequencies:

- the cells of the seed are divided by sum(seed);
- the cells of each margin i of the list target.data are divided by sum(target.data[[i]]).

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Bacharach, M. (1965). Estimating Nonnegative Matrices from Marginal Data. *International Economic Review* (Blackwell Publishing) 6 (3): 294-310.

Bishop, Y. M. M., Fienberg, S. E., Holland, P. W. (1975). *Discrete Multivariate Analysis: Theory and Practice.* MIT Press. ISBN 978-0-262-02113-5.

Deming, W. E., Stephan, F. F. (1940). On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. *Annals of Mathematical Statistics* 11 (4): 427-444.

Fienberg, S. E. (1970). An Iterative Procedure for Estimation in Contingency Tables. *Annals of Mathematical Statistics* 41 (3): 907-917.

Little, R. J., Wu, M. M. (1991) Models for contingency tables with known margins when target and sampled populations differ. *Journal of the American Statistical Association* 86 (413): 87-95.

Lang, J.B. (2004) Multinomial-Poisson homogeneous models for contingency tables. *Annals of Statistics* 32(1): 340-383.

Stephan, F. F. (1942). Iterative method of adjusting frequency tables when expected margins are known. *Annals of Mathematical Statistics* 13 (2): 166-178.

See Also

See the functions `Ipfp` and `ObtainModelEstimates` for more details about the estimation process.

`summary.mipfp` for summaries, `vcov.mipfp` for the (asymptotic) covariance of the estimates and `gof.estimates.mipfp` for testing if the seed agrees with the targets.

The genereric functions `print` and `coef`.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10 pct sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# applying one fitting method (ipfp)
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data)
print(r.ipfp)
```

---

expand *Expand a Table in a Data Frame*

Description

This function takes a multi-dimensionnal contingency table and expands it to a data frame containing individual records.

Usage

```
expand(x, ...)

## S3 method for class 'table'
expand(x, ...)
```

Arguments

x	An object of type `table` storing a N-dimensial contingency table.
...	Further arguments passed to or from other methods.

Value

A data frame of the individual records derived from `x`.

Note

The function is inspired from the "Cookbook for R".

It should also be noted that the cells od `x` are rounded before being expanded in a data frame.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Cookbook for R - http://www.cookbook-r.com/Manipulating_data/Converting_between_data_frames_and_contingency_tables/

See Also

`expand.grid` and `as.data.frame`.

Examples

```
# loading data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# create a contingency table
t <- table(spnamur.sub)
# expand the table to a data frame
t.df <- expand(t)
```

flat *Flatten a table, array or matrix*

Description

This function takes a multidimensional object and flattens it for a pretty printing. The row names are the concatenation of the original dimension names while the only column stores the initial data of the object.

Usage

```
## S3 method for class 'array'
flat(x, sep = ".", label = "value", l.names = 0, ...)

## S3 method for class 'table'
flat(x, sep = ".", label = "value", l.names = 0, ...)

## S3 method for class 'matrix'
flat(x, sep = ".", label = "value", l.names = 0, ...)
```

Arguments

`x`	An array, table or matrix.
`sep`	The separator used to concatenate the dimension names.
`label`	The name of the column storing the data.
`l.names`	If set to a value greater than 0, then the dimnames will be shorten to a length of `l.names` characters.
`...`	Not used.

Value

An array containing a flattened version of `x`.

Note

The function is inspired from the function `wrap.array` from the package `R.utils` written by Henrik Bengtsson.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

See Also

The function `wrap.array` from the `R.utils` package (https://cran.r-project.org/package=R.utils).

Examples

```
# loading the data and saving in a 3D-table
data(spnamur, package = "mipfp")
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
tab <- table(spnamur.sub)

# flattening the table
tab.flat <- flat(tab)
print(tab.flat)
```

`GetConfInt` *Computing confidence intervals for the estimated counts and probabilities (deprecated)*

Description

This function computes the (asymptotic) Wald confidence intervals at a given significance level for the results generated by `Ipfp` and `ObtainModelEstimates` (provided that their option `compute.cov` was set to `TRUE`).

Usage

```
GetConfInt(list.est, alpha = 0.05)
```

Arguments

`list.est`	A list produced either by `Ipfp` or `ObtainModelEstimates` containing the estimated counts and probabilities as well as their associated standard deviations.
`alpha`	Significance level of the confidence interval corresponding to the 100(1 - α)% confidence level.

Details

The confidence interval of the estimates $\hat{X}$, at significance level α is given by

$$\hat{X} \pm z\left(1 - \frac{\alpha}{2}\right) * \hat{\sigma}$$

where $\hat{\sigma}$ is the standart deviations of $\hat{X}$ and z is the inverse of the cumulative distribution function of the standard normal distribution.

Value

A list of matrices containing the upper and lower bounds for the estimated counts and probabilities.

`lower.x`	Lower bounds of the confidence interval for `list.est$x.hat`.
`upper.x`	Upper bounds of the confidence interval for `list.est$x.hat`.
`lower.p`	lower bounds of the confidence interval for `list.est$p.hat`.
`upper.p`	upper bounds of the confidence interval for `list.est$p.hat`.

Warning

Note: this function is deprecated, instead use confint.mipfp.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Smithson, M. (2002). *Confidence intervals*. Sage Publications.

See Also

Estimate, Ipfp and ObtainModelEstimates to generate the inputs for this function.

The S3 method confint.mipfp for object of class mipfp.

Examples

```
# true contingency (2-way) table
true.table <- array(c(43, 44, 9, 4), dim = c(2, 2))
# generation of sample, i.e. the seed to be updated
seed <- ceiling(true.table / 10)
# desired targets (margins)
target.row <- apply(true.table, 2, sum)
target.col <- apply(true.table, 1, sum)
# storing the margins in a list
target.data <- list(target.col, target.row)
# list of dimensions of each marginal constrain
target.list <- list(1, 2)
# calling the Ipfp function
res <- Ipfp(seed, target.list, target.data)
# addint the standart deviations to res (required by GetConfInt)
cov.res <- vcov(res, seed = seed, target.list = target.list,
                target.data = target.data)
res$p.hat.se <- cov.res$p.hat.se
res$x.hat.se <- cov.res$x.hat.se
# computing and printing the confidence intervals
print(GetConfInt(res))
```

GetLinInd — *Extracting the linearly independant columns from a matrix*

Description

Extracts the linearly dependant columns of matrix to obtain a matrix of full rank using QR decomposition.

Usage

```
GetLinInd(mat, tol = 1e-10)
```

Arguments

mat	The matrix possibly containing linearly dependant columns
tol	Rank estimation tolerance. Default is $1e^{-10}$.

Value

A list containing the new matrix and the index of the selected colums.

mat.li	A matrix made of the linearly independant columns of mat.
idx	The index of the selected columns.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Golub, G. H., Van Loan C. F. (2012) *Matrix Computations. Third Edition.* Johns Hopkins University Press.

See Also

qr.

Examples

```
# generation of a matrix with linearly dependant columns
A <- matrix(c(1, 2, 3,
              1, 2, 4,
              1, 2, 8), nrow = 3, ncol = 3, byrow = TRUE)

# extracting and printing the linearly independant columns
B <- GetLinInd(A)
print(B)
```

gof.estimates *Wald, Log-likelihood ratio and Person Chi-square statistics for mipfp object*

Description

This method computes three statistics to perform a test wheter the seed agrees with the target data. The statistics are the Wilk's log-likelihood ratio statistic, the Wald statistic and the Person Chi-square statistic.

The method also returns the associated degrees of freedom.

Usage

```
## S3 method for class 'mipfp'
gof.estimates(object, seed = NULL, target.data = NULL,
              target.list = NULL, replace.zeros = 1e-10, ...)
```

Arguments

`object`	The object of class `mipfp` containing.
`seed`	The seed used to compute the estimates (optional). If not provided, the method tries to determine the `seed` automatically.
`target.data`	A list containing the data of the target margins. Each component of the list is an array storing a margin. The list order must follow the one defined in `target.list`. Note that the cells of the arrays must be non-negative (and can even be NA if `method = ipfp`) (optional). If not provided, the method tries to dermine `target.data` automatically.
`target.list`	A list of the target margins provided in `target.data`. Each component of the list is an array whose cells indicates which dimension the corresponding margin relates to (optional). If not provided, the method tries to determine `target.list` automatically.
`replace.zeros`	If 0-cells are to be found, then they are replaced with this value.
`...`	Not used.

Details

The test is formally expressed as:

$$H_0 \;:\; h(\pi) = 0 \quad vs \quad H_1 \;:\; h(\pi) \neq 0$$

where π is the vector of the seed probabilities and $h(x) = A^T x - m$ with A and m being respectively the marginal matrix and the margins vector of the estimation problem.

The three statistics are then defined as:

- Wilk's log-likelihoold ratio
$$G^2 = 2 \sum x_i \ln \frac{\pi_i}{\hat{\pi}_i}$$

- Wald's statistic
$$W^2 = h(x)^T (H_x^T D_x H_x)^{-1} h(x)$$
- Pearson Chi-square
$$\chi^2 = (x - n\hat{\pi})^T D_{n\hat{\pi}}^{-1} (x - n\hat{\pi})$$

where x is the vectorization of the seed, $n = \sum x_i$, D_v is a diagonal matrix derived from the vector v and H denotes the Jacobian evaluated in $\hat{\pi}$ (the vector of the estimated probabilities) of the function $h(x)$.

The degrees of freedom for these statistics corresponds to the number of components in m.

Value

A list whose elements are detailed below.

G2	The Log-likelihood statistic.
W2	The Wald statistic.
X2	The Pearson chi-squared statistic.
stats.df	The degrees of freedom for the G2, W2 and X2 statistics.

Author(s)

Johan Barthelemy

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lang, J.B. (2004) Multinomial-Poisson homogeneous models for contingency tables. *Annals of Statistics* 32(1): 340-383.

See Also

Estimate function to create an object of class mipfp and to update an initial multidimensional array with respect to given constraints. summary.mipfp can also retrieve the statistics and their associated p-values.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
```

```
# creating the seed, a 10 pct sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
seed.table <- table(seed.df)
# applying one fitting method (ipfp)
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data)
# printing the G2, X2 and W2 statistics
print(gof.estimates(r.ipfp))
# alternative way (pretty printing, with p-values)
print(summary(r.ipfp)$stats.gof)
```

Ipfp *Multidimensional Iterative Proportional Fitting*

Description

This function implements the iterative proportional fitting (IPFP) procedure. This procedure updates an initial N-dimensional array (referred as the seed) with respect to given target marginal distributions. Those targets can also be multi-dimensional. This procedure is also able to estimate a (multi-dimensional) contingency table (encoded as an array) matching a given set of (multi-dimensional) margins. In that case, each cell of the seed must simply be set to 1.

The IPFP is also known as the RAS algorithm in economics and matrix raking or matrix scaling in computer science.

Usage

```
Ipfp(seed, target.list, target.data, print = FALSE, iter = 1000, tol = 1e-10,
     tol.margins = 1e-10, na.target = FALSE)
```

Arguments

`seed` The initial multi-dimensional array to be updated. Each cell must be non-negative.

`target.list` A list of dimensions of the marginal target constrains in `target.data`. Each component of the list is an array whose cells indicate which dimension the corresponding margin relates to.

`target.data` A list containing the data of the target marginal tables. Each component of the list is an array storing a margin. The list order must follow the ordering defined in `target.list`. Note that the cells of the arrays must be non-negative.

`print` Verbose parameter: if TRUE prints the current iteration number and the associated value of the stopping criterion. Default is FALSE.

`iter` Stopping criterion. The maximum number of iteration allowed; must be greater than 0. Default is 1000.

`tol` Stopping criterion. If the maximum absolute difference between two iteration is lower than the value specified by `tol`, then ipfp has reached convergence; must be greater than 0. Default is $1e^{-10}$.

`tol.margins`	Tolerance for the margins consistency. Default is $1e^{-10}$.
`na.target`	If set to TRUE, allows the targets to have NA cells. Note that in that particular case the margins consistency is not checked.

Value

A list containing the final updated array as well as other convergence informations.

`x.hat`	An array with the same dimension of `seed` whose margins match those specified in `target.list`.
`p.hat`	An array with the same dimension of `x.hat` containing the updated cell probabilities, i.e. `x.hat / sum(x.hat)`.
`evol.stp.crit`	The evolution of the stopping criterion over the iterations.
`conv`	A boolean indicating whether the algorithm converged to a solution.
`error.margins`	A list returning, for each margin, the absolute maximum deviation between the desired and generated margin.
`method`	The selected method for estimation (here it will always be `ipfpf`).
`call`	The matched call.

Note

It is important to note that if the margins given in `target.list` are not consistent (i.e. the sums of their cells are not equals), the input data is then normalised by considering probabilities instead of frequencies:

- the cells of the seed are divided by `sum(seed)`;
- the cells of each margin `i` of the list `target.data` are divided by `sum(target.data[[i]])`.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

References

Bacharach, M. (1965). Estimating Nonnegative Matrices from Marginal Data. *International Economic Review* (Blackwell Publishing) 6 (3): 294-310.

Bishop, Y. M. M., Fienberg, S. E., Holland, P. W. (1975). *Discrete Multivariate Analysis: Theory and Practice.* MIT Press. ISBN 978-0-262-02113-5.

Deming, W. E., Stephan, F. F. (1940). On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. *Annals of Mathematical Statistics* 11 (4): 427-444.

Fienberg, S. E. (1970). An Iterative Procedure for Estimation in Contingency Tables. *Annals of Mathematical Statistics* 41 (3): 907-917.

Stephan, F. F. (1942). Iterative method of adjusting frequency tables when expected margins are known. *Annals of Mathematical Statistics* 13 (2): 166-178.

See Also

The documentation of IpfpCov provide details on the the covariance matrices determination.

ObtainModelEstimates for alternatives to the IPFP.

Examples

```
# Example 1: 2-way table (V1,V2) of dim=(2,2)
# generating an intial 2-way table to be updated
seed.2d <- array(1,dim=c(2,2))
# desired targets (margins) : V1 and V2
target.row <- c(50,50)
target.col <- c(30,70)
# storing the margins in a list
tgt.data.2d <- list(target.col, target.row)
# list of dimensions of each marginal constrain
tgt.list.2d <- list(1,2)
# calling the Ipfp function
res.2d <- Ipfp(seed.2d, tgt.list.2d, tgt.data.2d)

# Example 2: 3-way table (V1,V2,V3) of dim=(2,4,2)
# seed
seed.3d <- array(1,c(2,4,2))
seed.3d[1,1,1] <- 4
seed.3d[1,3,1] <- 10
seed.3d[1,4,2] <- 6
# desired targets (margins) : V1 and (V2,V3)
target.V1 <- c(50, 16)
target.V2.V3 <- array(4, dim=c(4,2))
target.V2.V3[1,1] <- 10
target.V2.V3[3,1] <- 22
target.V2.V3[4,2] <- 14
# list of dimensions of each marginal constrain
tgt.data.3d <- list(target.V1, target.V2.V3)
# storing the description of target data in a list
tgt.list.3d <- list( 1, c(2,3) )
# calling the Ipfp function
res.3d <- Ipfp(seed.3d, tgt.list.3d, tgt.data.3d, iter=50, print=TRUE, tol=1e-5)

# Example 3: 2-way table (V1,V2) of dim=(2,3) with missing values in the targets
# generating an intial 2-way table to be updated
seed.2d.na <- array(1,dim=c(2,3))
# desired targets (margins) : V1 and V2
target.row.na <- c(40,60)
target.col.na <- c(NA,10,NA)
# storing the margins in a list
tgt.data.2d.na <- list(target.row.na, target.col.na)
# storing the description of target data in a list
tgt.list.2d.na <- list(1,2)
# calling the Ipfp function
res.2d.na <- Ipfp(seed.2d.na, tgt.list.2d.na, tgt.data.2d.na, na.target=TRUE)
```

IpfpCov *Covariance matrix of the estimators produced by Ipfp (deprecated)*

Description

This function determines the (asymptotic) covariance matrix of the estimates produced by the iterative proportional fitting procedure using the formula designed by Little and Wu (1991).

Usage

```
IpfpCov(estimate, seed, target.list, replace.zeros = 1e-10)
```

Arguments

`estimate`	The array of estimates produced by the `Ipfp` function.
`seed`	The intial array (seed) that was updated by the `Ipfp` function.
`target.list`	A list of dimensions of the marginal target constrains. Each component of the list is an array whose cells indicate which dimension the corresponding margin relates to.
`replace.zeros`	If a cell of the `estimate` or the `seed` has a value equals to 0, then it is replaced with this value. Default is 1e-10.

Details

The asymptotic covariance matrix of the estimates produced by the iterative proportional fitting procedure has the form (Little and Wu, 1991)

$$K(K^T D1^{-1}K)^{-1}K^T D2^{-1}K(K^T D1^{-1}K)^{-1}K^T$$

where

- `K` is the orthogonal complement of the marginal matrix, i.e. the matrix required to obtain the marginal frequencies;
- `D1` is a diagonal matrix of the estimates probabilities;
- `D2` is a diagonal matrix of the seed probabilities.

Value

A matrix of dimension `length(estimate)` x `length(estimate)` of the asymptotic variance of the proportion estimates produced by `Ipfp`.

Warning

Note: this function is deprecated, instead use `vcov.mipfp`.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Little, R. J., Wu, M. M. (1991) Models for contingency tables with known margins when target and seed populations differ. *Journal of the American Statistical Association* 86 (413): 87-95.

See Also

Ipfp function to update an initial multidimensional array with respect to given constraints.

Examples

```
# true contingency (2-way) table
true.table <- array(c(43, 44, 9, 4), dim = c(2, 2))
# generation of sample, i.e. the seed to be updated
seed <- ceiling(true.table / 10)
# desired targets (margins)
target.row <- apply(true.table, 2, sum)
target.col <- apply(true.table, 1, sum)
# storing the margins in a list
target.data <- list(target.col, target.row)
# list of dimensions of each marginal constrain
target.list <- list(1, 2)
# calling the Ipfp function
res <- Ipfp(seed, target.list, target.data)
# computation of the covariance matrix of the produced estimated probabilities
res.cov <- IpfpCov(res$x.hat, seed, target.list)
# 0.95 level confidence interval of the estimates
n <- sum(res$x.hat)
# ... lower bound
ci.lb <- Array2Vector(res$x.hat) - 1.96 * sqrt(n * diag(res.cov))
# ... upperbound
ci.ub <- Array2Vector(res$x.hat) + 1.96 * sqrt(n * diag(res.cov))
```

ObtainModelEstimates *Estimating a contingency table using model-based approaches*

Description

This function provides several alternative estimating methods to the IPFP when estimating a multiway table subject to known constrains/totals: maximum likelihood method (ML), minimum chi-squared (CHI2) and weighted least squares (WLSQ). Note that the resulting estimators are probabilities.

The covariance matrix of the estimated proportions (as defined by Little and Wu, 1991) are also provided. Also in the case of the ML method, the covariance matrix defined by Lang (2004) is also returned.

Usage

```
ObtainModelEstimates(seed, target.list, target.data, method="ml",
                     tol.margins = 1e-10, replace.zeros = 1e-10, ...)
```

Arguments

- `seed` — The initial multi-dimensional array to be updated. Each cell must be non-negative.
- `target.list` — A list of the target margins provided in `target.data`. Each component of the list is an array whose cells indicates which dimension the corresponding margin relates to.
- `target.data` — A list containing the data of the target margins. Each component of the list is an array storing a margin. The list order must follow the one defined in `target.list`. Note that the cells of the arrays must be non-negative.
- `method` — Determine the model to be used for estimating the contingency table. By default the method is `ml` (maximum likelihood); other options available are `chi2` (minimum chi-squared) and `lsq` (least squares).
- `tol.margins` — Tolerance for the margins consistency. Default is $1e^{-10}$.
- `replace.zeros` — Constant that is added to zero cell found in the seed, as procedures require strictly positive cells. Default value is $1e^{-10}$.
- `...` — Additional parameters that can be passed to control the optimisation process (see solnp from the package Rsolnp).

Value

A list containing the final estimated table as well as the covariance matrix of the estimated proportion and other convergence informations.

- `x.hat` — Array of the estimated table frequencies.
- `p.hat` — Array of the estimated table probabilities.
- `error.margins` — For each list element of `target.data`, `check.margins` shows the maximum absolute deviation between the element and the corresponding estimated margin. Note that the deviations should approximate zero, otherwise the target margins are not met.
- `solnp.res` — The estimation process uses the `solnp` optimisation function from the R package Rsolnp and `solnp.res` is the corresponding object returned by the solver.
- `conv` — A boolean indicating whether the algorithm converged to a solution.
- `method` — The selected method for estimation.
- `call` — The matched call.

Note

It is important to note that if the margins given in `target.list` are not consistent (i.e. the sums of their cells are not equals), the input data is then normalised by considering probabilities instead of frequencies:

- the cells of the seed are divided by `sum(seed)`;
- the cells of each margin `i` of the list `target.data` are divided by `sum(target.data[[i]])`.

Author(s)

Thomas Suesse

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lang, J.B. (2004) Multinomial-Poisson homogeneous models for contingency tables. *Annals of Statistics* 32(1): 340-383.

Little, R. J., Wu, M. M. (1991) Models for contingency tables with known margins when target and sampled populations differ. *Journal of the American Statistical Association* 86 (413): 87-95.

See Also

`solnp` function documentation of the package `Rsolnp` for the details of the `solnp.res` object returned by the function.

Examples

```
# set-up an initial 3-way table of dimension (2 x 2 x 2)
seed <- Vector2Array(c(80, 60, 20, 20, 40, 35, 35, 30), dim = c(c(2, 2, 2)))

# building target margins
margins12 <- c(2000, 1000, 1500, 1800)
margins12.array <- Vector2Array(margins12, dim=c(2, 2))
margins3 <- c(4000,2300)
margins3.array <- Vector2Array(margins3, dim = 2)
target.list <- list(c(1, 2), 3)
target.data <- list(margins12.array, margins3.array)

# estimating the new contingency table using the ml method
results.ml <- ObtainModelEstimates(seed, target.list, target.data,
                                   compute.cov = TRUE)
print(results.ml)

# estimating the new contingency table using the chi2 method
results.chi2 <- ObtainModelEstimates(seed, target.list, target.data,
                                     method = "chi2", compute.cov = TRUE)
print(results.chi2)

# estimating the new contingency table using the lsq method
results.lsq <- ObtainModelEstimates(seed, target.list, target.data,
                                    method = "lsq", compute.cov = TRUE)
print(results.lsq)
```

`ObtainMultBinaryDist` *Generating a multivariate Bernoulli joint-distribution*

Description

This function applies the IPFP procedure to obtain a joint distribution of K multivariate binary (Bernoulli) variables X_1, ..., X_K.

It requires as input the odds ratio or alternatively the correlation as a measure of association between all the binary variables and a vector of marginal probabilities.

This function is useful when one wants to simulate and draw from a multivariate binary distribution when only first order (marginal probabilities) and second order moments (correlation or odds ratio) are available.

Usage

```
ObtainMultBinaryDist(odds = NULL, corr = NULL, marg.probs, ...)
```

Arguments

`odds`	A $K \times K$ matrix where the i-th row and the j-th column represents the Odds ratio between variables i and j. Must be provided if `corr` is not.
`corr`	A $K \times K$ matrix where the i-th row and the j-th column represents the correlation between variables i and j. Must be provided if `odds` is not.
`marg.probs`	A vector with K elements of marginal probabilities where the i-th entry refers to $P(X_i = 1)$.
`...`	Additional arguments that can be passed to the `Ipfp` function such as `tol`, `iter`, `print` and `compute.cov`.

Value

A list whose elements are mainly determined by the `Ipfp` function.

`joint.proba`	The resulting multivariate joint-probabilities (from `Ipfp`).
`stp.crit`	The final value of the `Ipfp` stopping criterion.
`conv`	Boolean indicating whether the `Ipfp` algorithm converged to a solution.
`check.margins`	A list returning, for each margin, the absolute maximum deviation between the desired and generated margin. Ideally the elements should approximate 0 (from `Ipfp`).
`label`	The names of the variables.

Note

It is important to note that either the odds ratio defined in `odds` or the correlations described in `corr` must be provided.

Author(s)

Thomas Suesse

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

`Ipfp` for the function used to estimate the distribution; `RMultBinary` to simulate the estimated joint-distribution; `Corr2Odds` and `Odds2Corr` to convert odds ratio to correlation and conversely.

Examples

```
# initial odds ratios from Qaqish et al. (2012)
or <- matrix(c(Inf, 0.281, 2.214, 2.214,
               0.281, Inf, 2.214, 2.214,
               2.214, 2.214, Inf, 2.185,
               2.214, 2.214, 2.185, Inf), nrow = 4, ncol = 4, byrow = TRUE)
rownames(or) <- colnames(or) <- c("Parent1", "Parent2", "Sibling1", "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# estimating the joint-distribution
p.joint <- ObtainMultBinaryDist(odds = or, corr = NULL, marg.probs = p)
print(p.joint$joint.proba)

# obtain identical solution when providing correlation
corr <- Odds2Corr(odds = or, marg.probs = p)$corr
p.joint.alt <- ObtainMultBinaryDist(corr = corr, marg.probs = p)

# checking if the results are truly identicals
diff <- sum(abs(p.joint.alt$joint.proba - p.joint$joint.proba))
cat('Sum of the absolute deviations: ', diff, '\n')
```

`Odds2Corr` *Converting odds ratio to correlation*

Description

For K binary (Bernoulli) random variables X_1, ..., X_K, this function transforms the odds ratios measure of association O_{ij} between every pair (X_i, X_j) to the correlation C_{ij} where

$$C_{ij} = \frac{cov(X_i, X_j)}{\sqrt{var(X_i) * var(X_j)}}$$

and

$$O_{ij} = \frac{P(X_i = 1, X_j = 1) * P(X_i = 0, X_j = 0)}{P(X_i = 1, X_j = 0) * P(X_i = 0, X_j = 1)}.$$

Usage

```
Odds2Corr(odds, marg.probs)
```

Arguments

`odds`	A $K \times K$ matrix where the i-th row and the j-th column represents the odds ratio O_{ij} between variables i and j.
`marg.probs`	A vector with K elements of marginal probabilities where the i-th entry refers to $P(X_i = 1)$.

Value

The function return a list with the correlations and the pairwise probabilities.

`corr`	A matrix of the same dimension as `odds` containing the correlations
`pair.proba`	A matrix of the same dimension as `odds` containing the pairwise probabilities.

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <`johan@uow.edu.au`>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

`Corr2Odds` for converting correlation to odds ratio.

Examples

```
# from Qaqish et al. (2012)
or <- matrix(c(Inf, 0.281, 2.214, 2.214,
               0.281, Inf, 2.214, 2.214,
               2.214, 2.214, Inf, 2.185,
               2.214, 2.214, 2.185, Inf), nrow = 4, ncol = 4, byrow = TRUE)
rownames(or) <- colnames(or) <- c("Parent1", "Parent2", "Sibling1", "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# converting odds ratio to correlation
corr <- Odds2Corr(odds = or, marg.probs = p)
print(corr)
```

`Odds2PairProbs` *Converting odds ratio to pairwise probability*

Description

For K binary (Bernoulli) random variables X_1, ..., X_K, this function transforms the odds ratios measure of association O_{ij} between every pair (X_i, X_j) to the pairwise probability $P(X_i = 1, X_j = 1)$, where O_{ij} is defined as

$$O_{ij} = \frac{P(X_i = 1, X_j = 1) * P(X_i = 0, X_j = 0)}{P(X_i = 1, X_j = 0) * P(X_i = 0, X_j = 1)}.$$

Usage

```
Odds2PairProbs(odds, marg.probs)
```

Arguments

`odds`	A $K \times K$ matrix where the i-th row and the j-th column represents the odds ratio O_{ij} between variables i and j.
`marg.probs`	A vector with K elements of marginal probabilities where the i-th entry refers to $P(X_i = 1)$.

Value

A matrix of the same dimension as odds containing the pairwise probabilities

Note

If we denote $P(X_i = 1, X_j = 1)$ by h_{ij}, and $P(X_i = 1)$ by p_i, then it can be shown that

$$O_{ij} = \frac{h_{ij} * (1 - p_i - p_j + h_{ij})}{((p_i - h_{ij}) * (p_j - h_{ij}))}$$

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

Corr2PairProbs for converting the correlation to pairwise probability.

Examples

```
# from Qaqish et al. (2012)
or <- matrix(c(Inf, 0.281, 2.214, 2.214,
               0.281, Inf, 2.214, 2.214,
               2.214, 2.214, Inf, 2.185,
               2.214, 2.214, 2.185, Inf), nrow = 4, ncol = 4, byrow = TRUE)
rownames(or) <- colnames(or) <- c("Parent1", "Parent2", "Sibling1", "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# getting the pairwise probabilities
pp <- Odds2PairProbs(odds = or, marg.probs = p)
print(pp)
```

Qaqish *Qaqish*

Description

The data set provides the odds ratios and correlations as measures of associations of the binary outcome impaired pulmonary function for a family of four with two parents and two siblings.

These correlations and odds ratios are obtained from Qaqish et al. (2012) based on a regression analysis of a common data set of parents and siblings with chronic obstructive pulmonary disease and their controls.

Usage

```
data(Qaqish)
```

Format

A list `Qaqish` containing 2 elements:

1. `cr` : the correlation matrix;
2. `or` : the odd ratios matrix.

Source

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

Examples

```
data(Qaqish)
print(Qaqish$or)
print(Qaqish$cr)
```

`RMultBinary` *Simulating a multivariate Bernoulli distribution*

Description

This function generates a sample from a multinomial distribution of K dependent binary (Bernoulli) variables $(X_1, X_2, ..., X_K)$ defined by an array (of 2^K cells) detailing the joint-probabilities.

Usage

```
RMultBinary(n = 1, mult.bin.dist, target.values = NULL)
```

Arguments

`n`	Desired sample size. Default = 1.
`mult.bin.dist`	A list describing the multivariate binary distribution. It can be generated by the `ObtainMultBinaryDist` function. The list contains at least the element `joint.proba`, an array detailing the joint-probabilities of the K binary variables. The array has K dimensions of size 2, referring to the 2 possible outcomes of the considered variable. Hence, the total number of elements is 2^K. Additionnaly the list can also provides the element `var.label`, a list containing the names of the K variables.
`target.values`	A list describing the possibles outcomes of each binary variable, for instance {1, 2}. Default = {0, 1}.

Value

A list whose elements are detailed herehunder.

`binary.sequences`
: The generated $K \times n$ random sequence.

`possible.binary.sequences`
: The possible binary sequences, i.e. the domain.

`chosen.random.index`
: The index of the random draws in the domain.

Author(s)

Thomas Suesse

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lee, A.J. (1993). Generating Random Binary Deviates Having Fixed Marginal Distributions and Specified Degrees of Association. *The American Statistician* 47 (3): 209-215.

Qaqish, B. F., Zink, R. C., and Preisser, J. S. (2012). Orthogonalized residuals for estimation of marginally specified association parameters in multivariate binary data. *Scandinavian Journal of Statistics* 39, 515-527.

See Also

`ObtainMultBinaryDist` for estimating the joint-distribution required by this function.

Examples

```
# from Qaqish et al. (2012)
or <- matrix(c(Inf, 0.281, 2.214, 2.214,
               0.281, Inf, 2.214, 2.214,
               2.214, 2.214, Inf, 2.185,
               2.214, 2.214, 2.185, Inf), nrow = 4, ncol = 4, byrow = TRUE)
rownames(or) <- colnames(or) <- c("Parent1", "Parent2", "Sibling1", "Sibling2")

# hypothetical marginal probabilities
p <- c(0.2, 0.4, 0.6, 0.8)

# estimating the joint-distribution
p.joint <- ObtainMultBinaryDist(odds = or, marg.probs = p)

# simulating 100,000 draws from the obtained joint-distribution
y.sim <- RMultBinary(n = 1e5, mult.bin.dist = p.joint)$binary.sequences

# checking results
cat('dim y.sim =', dim(y.sim)[1], 'x', dim(y.sim)[2], '\n')
cat('Estimated marginal probs from simulated data\n')
apply(y.sim,2,mean)
cat('True probabilities\n')
```

```
print(p)
cat('Estimated correlation from simulated data\n')
cor(y.sim)
cat('True correlation\n')
Odds2Corr(or,p)$corr

# generating binary outcomes with outcome different than 0, 1
RMultBinary(n = 10, mult.bin.dist = p.joint,
            target.values = list(c("A", "B"), c(0, 1), c(1, 2), c(100, 101)))
```

---

spnamur *Synthetic population of Namur (Belgium)*

Description

This data drame contains a synthetic population of individuals for Belgian city of Namur. The attributes details the gender, age class, socio-professional status, education level and driving license ownership of every synthetic individual.

Usage

```
data(spnamur)
```

Format

A data frame detailing the synthetic individuals whose columns are described in the Table below.

Attribute	**Values (levels)**
Household.type	C (couple); F (family with children); I (isolated); N (non family)
Gender	F (female); H (male)
Prof.status	A (active); E (student); I (inactive)
Education.level	O (none); P (primary); S (high school); U (higher education)
Driving.license	O (no); P (yes)
Age.class	0 (0-5); 1 (6-17); 2 (18-39); 3 (40-59); 4 (60+)

Source

VirtualBelgium - http://virtualbelgium.sourceforge.net

References

Barthelemy, J. and Toint, P.L. (2013) Synthetic population generation without a sample *Transportation Science* 47 (2): 266-279

Examples

```
data(spnamur)
```

```
# generating the contingency table of the synthetic population
table(spnamur)
```

---

summary.mipfp *Summarizing objects of class mipfp*

Description

Summary method for class `mipfp`.

Usage

```
## S3 method for class 'mipfp'
summary(object, cov.method = "delta", prop = FALSE,
        target.list = NULL, l.names = 0, ...)

## S3 method for class 'summary.mipfp'
print(x, ...)
```

Arguments

`object`	An object of class `mipfp`, usually a result of a call to Estimate
`x`	An object of class `summary.mipfp`, usually a result of a call to `summary.mipfp`.
`cov.method`	Indicates which method to use to compute the covariance. Possible values are Delta (`delta`, default) or Lang (`lang`).
`prop`	If set to `FALSE` (the default), the results return counts, probabilities otherwise.
`target.list`	The list of the dimensions of the targets used by for the estimation process (see Estimate for more details).
`l.names`	If set to a value greater than 0, then the names of the categories will be shorten to a length of `l.names` characters.
`...`	Further arguments passed to the underlying `print` and `flat` method, or from other methods.

Details

The function `summary.mipfp` compute and returns a list of summary statistics of the estimates (covariance, t-statistics, goodness-of-fit statistics, associated degrees of freedom).

Value

The function `summary.mipfp` returns an object of class `summary.mipfp` having the following components:

`call`	A `call` object in which all the specified arguments are given by their full names.
`conv`	A Boolean indicating if the specified method converged to a solution (`TRUE`) or not (`FALSE`).

method	The method used to generate estimates.
df	Degrees of freedom of the estimates.
estimates	Estimates generated by the selected method with standard deviations and associated t- and p-values.
error.margins	A list returning, for each margin, the absolute maximum deviation between the desired and generated margin.
vcov	A covariance matrix of the estimates (last index move fastest) computed using the method specified in cov.method.
tab.gof	A table containing the Log-likelihood (G2), Wald (W2) and Pearson chi-squared (X2) statistics with their associated p-values.
stats.df	Degrees of freedom for the G2, W2 and X2 statistics.
dim.names	Original dimension names of the estimated table.
l.names	The value of the parameter l.names.

Note

When using print for printing the resulting mipfp object, you can also have a look at the options of the method flat.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

See Also

The estimation function Estimate.

The function coef.mipfp to extract the estimates.

gof.estimates for the computation of the G2, W2 and X2 statistics.

vcov.mipfp for the details of the covariance computation.

Examples

```
# loading the data
data(spnamur, package = "mipfp")
# subsetting the data frame, keeping only the first 3 variables
spnamur.sub <- subset(spnamur, select = Household.type:Prof.status)
# true table
true.table <- table(spnamur.sub)
# extracting the margins
tgt.v1        <- apply(true.table, 1, sum)
tgt.v1.v2     <- apply(true.table, c(1,2), sum)
tgt.v2.v3     <- apply(true.table, c(2,3), sum)
tgt.list.dims <- list(1, c(1,2), c(2,3))
tgt.data      <- list(tgt.v1, tgt.v1.v2, tgt.v2.v3)
# creating the seed, a 10 pct sample of spnamur
seed.df <- spnamur.sub[sample(nrow(spnamur), round(0.10*nrow(spnamur))), ]
```

```
seed.table <- table(seed.df)
# applying the different fitting methods
r.ipfp <- Estimate(seed=seed.table, target.list=tgt.list.dims,
                   target.data = tgt.data)
# printing the summary
print(summary(r.ipfp))
```

---

vcov.mipfp *Calculate variance-covariance matrix for mipfp objects*

Description

This function determines the (asymptotic) covariance matrix of the estimates in an mipfp object using either the Delta formula designed by Little and Wu (1991) or Lang's formula (2004).

Usage

```
## S3 method for class 'mipfp'
vcov(object, method.cov = "delta", seed = NULL,
     target.data = NULL, target.list = NULL, replace.zeros = 1e-10, ...)
```

Arguments

object	An object of class mipfp.
method.cov	Select the method to use for the computation of the covariance. The available methods are delta and lang.
seed	The initial multi-dimensional array used to create object (optional).
target.data	A list containing the data of the target margins used to create object. Each component of the list is an array storing a margin. The list order must follow the one defined in target.list (optional).
target.list	A list of the target margins used to create object function. Each component of the list is an array whose cells indicates which dimension the corresponding margin relates to (optional).
replace.zeros	If 0-cells are to be found, then their values are replaced with this value.
...	Not used.

Details

The asymptotic covariance matrix of the estimates probabilities using Delta's formula has the form (Little and Wu, 1991)

$$K(K^T D1^{-1} K)^{-1} K^T D2^{-1} K(K^T D1^{-1} K)^{-1} K^T$$

where

- K is the orthogonal complement of the marginal matrix, i.e. the matrix A required to obtain the marginal frequencies m;

- D1 and D2 are two diagonal matrices whose components depends on the estimation process used to generate `object`.

If the estimation process has been done using

- `ipfp` then $diag(D1) = \hat{p}$ and $diag(D2) = p_*$;
- `ml` then $diag(D1) = \frac{\hat{p}^2}{p_*}$ and $diag(D2) = diag(D1)$;
- `chi2` then $diag(D1) = \frac{\hat{p}^4}{p_*^3}$ and $diag(D2) = diag(D1)$;
- `lsq` then $diag(D1) = p_*$ and $diag(D2) = \frac{p_*3}{\hat{p}^2}$;

where $\hat{p}$ is the vector of estimated probabilities and p_* is the vector of the seed probabilities.

Using Lang's formula (2004), the covariance matrix becomes

$$\frac{1}{N}\left(D - \hat{p}\hat{p}^T - DH(H^T DH)^{-1} H^T D\right)$$

where

- D is a diagonal matrix of the estimated probabilities $\hat{p}$;
- H denotes the Jacobian evaluated in $\hat{p}$ of the function $h(p) = A^T p - m$.

Value

A list with the following components:

`x.hat.cov`	A covariance matrix of the estimated counts (last index move fastest) computed using the method specified in `cov.method`.
`p.hat.cov`	A covariance matrix of the estimated probabilities (last index move fastest) computed using the method specified in `cov.method`.
`x.hat.se`	The standard deviation of the estimated counts (last index move fastest) computed using the method specified in `cov.method`.
`p.hat.se`	The standard deviation of the estimated probabilities (last index move fastest) computed using the method specified in `cov.method`.
`df`	Degrees of freedom of the estimates.
`method.cov`	The method used to compute the covariance matrix.

Author(s)

Johan Barthelemy.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

References

Lang, J.B. (2004) Multinomial-Poisson homogeneous models for contingency tables. *Annals of Statistics* 32(1): 340-383.

Little, R. J., Wu, M. M. (1991) Models for contingency tables with known margins when target and seed populations differ. *Journal of the American Statistical Association* 86 (413): 87-95.

See Also

Estimate function to create an object of class mipfp and to update an initial multidimensional array with respect to given constraints.

Examples

```
# true contingency (2-way) table
true.table <- array(c(43, 44, 9, 4), dim = c(2, 2))
# generation of sample, i.e. the seed to be updated
seed <- ceiling(true.table / 10)
# desired targets (margins)
target.row <- apply(true.table, 2, sum)
target.col <- apply(true.table, 1, sum)
# storing the margins in a list
target.data <- list(target.col, target.row)
# list of dimensions of each marginal constrain
target.list <- list(1, 2)
# calling the Estimate function
res <- Estimate(seed, target.list, target.data)
# printing the variance-covariance matrix
print(vcov(res))
```

Vector2Array *Transforming a vector to an array*

Description

Transform a vector into a multidimensional array. The transformation is done assuming that the last index of the array moves fastest. For instance, the relation between a vector v of lenght 8 and an array a of dimensions (2,2,2) is defined by $v = (a_{111}, a_{112}, a_{113}, a_{121}, a_{122}, \ldots, a_{333}$.

Usage

```
Vector2Array(vect, dim.out)
```

Arguments

vect	The vector of length one or more to be transformed into an array.
dim.out	The dimension attribute for the array to be created, that is an integer vector of length one or more giving the maximal indices in each dimension.

Value

An array of dimensions given by dim.out filled with the data from the input vector vec.

Author(s)

Thomas Suesse.

Maintainer: Johan Barthelemy <johan@uow.edu.au>.

See Also

The inverse transformation is performed with the function `Array2Vector`.

Examples

```
# generate a vector [1,2,...,27]
v <- seq(1:27)
# transform it into an array of dimension (3,3,3)
a <- Vector2Array(v,c(3,3,3))
```

Index